

Data Structures & Algorithms in Python

1st Edition Errata

Updated: 5 May 2023

This document contains all the corrections to the first edition of *Data Structures & Algorithms in Python*. Check the <https://datastructures.live> website for updates to this document.

By Chapter

- Chapter 7
- Chapter 9
- Chapter 10
- Chapter 12
- Chapter 13
- Chapter 14
- Chapter 15

Chapter 7

Chapter 7, "The Basic Quicksort Algorithm", Page 302, Listing 7-4

In the definition of `quicksort_sketch()` the call to `self.partition()` should be modified as follows:

```
    hipart = self.partition( # Partition array around the key
        key(self.get(pivot)), # of the item at the pivot index and
        lo, hi, key)         # record where high part starts
```

Chapter 9

Chapter 9, "Python Code for a 2-3-4 Tree" > "The Tree234 Class" > "Splitting Full Nodes", Page 420, Listing 9-5

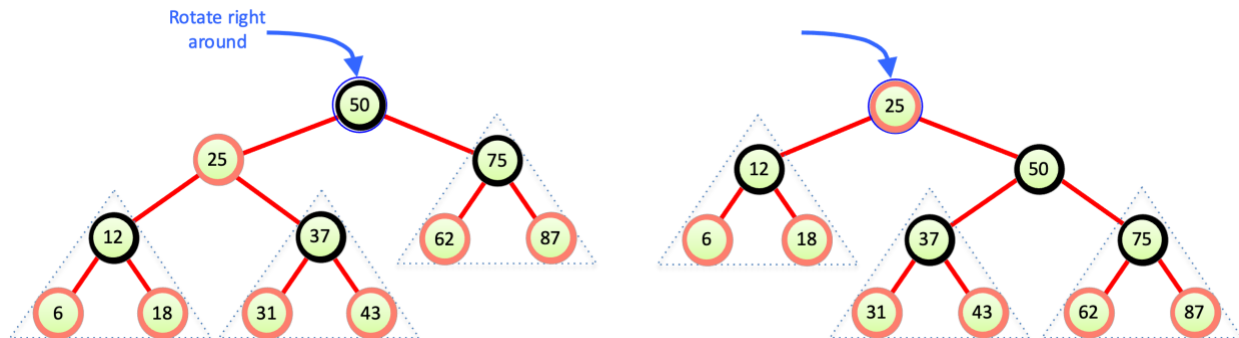
The code for the `insert()` method has an incorrect comment for the test `if p is self`. It should be as shown here:

```
    if p is self:          # Check if the parent is the tree
```

Chapter 10

Chapter 10, "Rotations in Red-Black Trees" > "Subtrees on the move", Page 498, Figure 10-22

In the lefthand tree (the one before the rotation operation), the key in the leftmost node at level 2 should be 12 instead of 71, as shown here:



Chapter 10, "Inserting a New Node" > "Color Swaps on the way down", Page 500, before Figure 10-23

In the description of Figure 10-23, the text should say

“... let’s call the node at the top of the triangle, the one that’s **black** before the swap, P for parent”

instead of

“... let’s call the node at the top of the triangle, the one that’s **red** before the swap, P for parent.”

Chapter 12

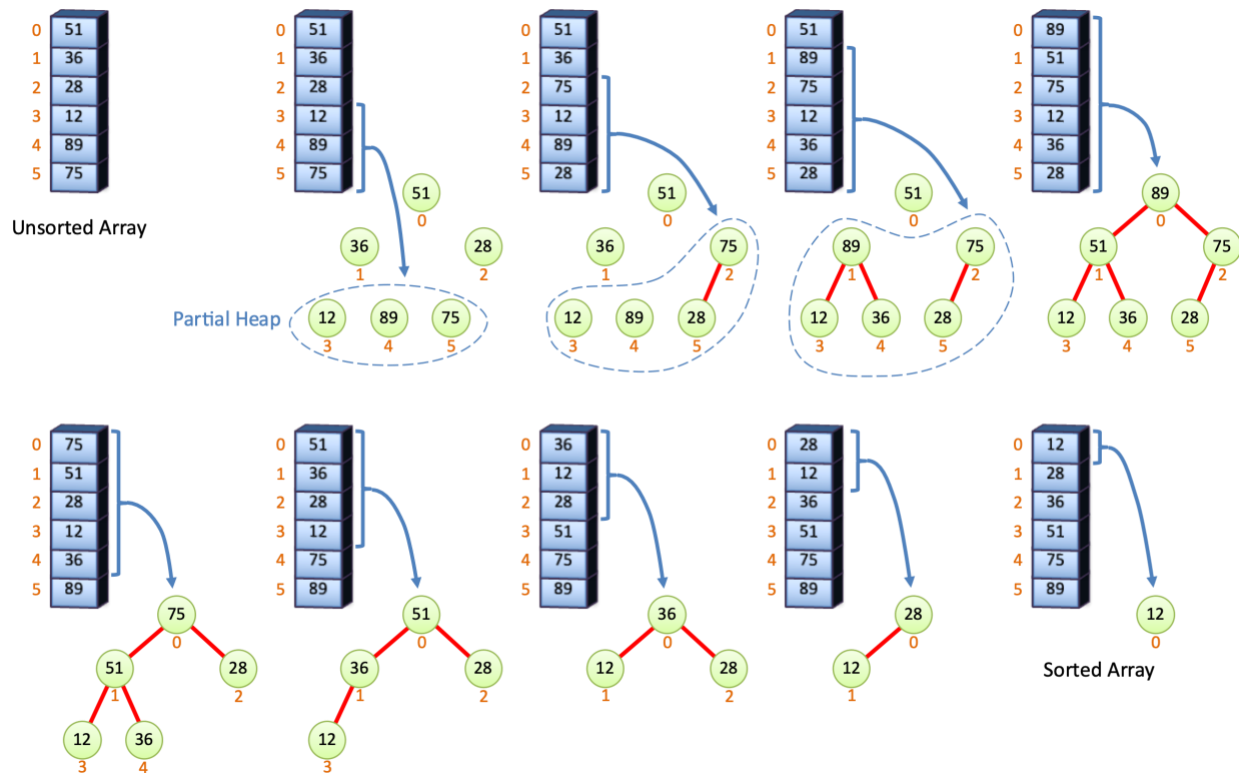
Chapter 12, "Grids" > "Does the Query Circle Intersect a Grid Cell", Page 628, 2nd paragraph

The paragraph should begin, “Figure 12-13 illustrates five query circle sample **scenarios** **which** might occur when determining whether any part of a query circle falls within the center grid cell.”

Chapter 13

Chapter 13, "Heapsort" > "Using the Same Array", Page 690, Figure 13-14

In Figure 13-14, the tree heap in the bottom left of the figure should show 51 instead of 59 so it looks like:



Chapter 14

Chapter 14, "Connectivity in Directed Graphs" > "Transitive Closure and Warshall's Algorithm", Page 755, 2nd paragraph in the description of "Row A"

The description should be

"In the `_adjMat` of Figure 14-21, column A has ones (1s) in rows B and E.

The first indicates there's an edge from B to A."

instead of

"In the `_adjMat` of Figure 14-21, there's only one 1 in column A: at row B.

It says there's an edge from B to A."

The last sentence should be

"That's what you expect in adjacency matrices because only pseudographs allow vertices to have edges to themselves."

instead of

"That's what you expect in adjacency matrices because only pseudographs to allow vertices to have edges to themselves."

Chapter 15

The last two sentences should be

"The `weight()` function is shown at the end of Listing 15-2 and returns the negative of the edge weight, which is the second element of the tuple. Using the negative value puts the lowest weight edges at the top of the heap."

instead of

"The `weight()` function is shown at the end of Listing 15-2. It returns the second element of the tuple."

The code should be

```
class WeightedGraph(object): # A graph containing vertices and edges
...

    def minimumSpanningTree( # Compute a spanning tree minimizing edge
        self, n):           # weight starting at vertex n
        self.validIndex(n)   # Check that vertex n is valid
        tree = WeightedGraph() # Initial MST is an empty weighted graph
        nVerts = self.nVertices() # Number of vertices
        vMap = [None] * nVerts # Array to map vertex indices into MST
        edges = Heap(         # Use heap for explored edges
            key=weight,        # Store (A, B) vertex pair & weight in
            )                  # each heap item, order by negative weight
        vMap[n] = 0           # Map start vertex into MST
        tree.addVertex(self.getVertex(n)) # Copy vertex n into MST
        while tree.nVertices() < nVerts: # Loop until all verts mapped
            for vertex in self.adjacentVertices(n): # For all adjacent
                if not vMap[vertex]: # vertices that are not mapped,
                    edges.insert( # put weighted edges in heap
                        ((n, vertex), self.edgeWeight(n, vertex)))
            edge, w = (         # Get first edge and weight, if one exists
                (None, 0) if edges.isEmpty() else edges.remove())
            while (not edges.isEmpty() and # While there are more edges
                vMap[edge[1]] is not None): # and edge within MST,
                edge, w = edges.remove()    # go on to next edge
            if (edge is None or # If we didn't find an edge or it goes
                vMap[edge[1]] is not None): # to a mapped vertex
                break           # there are no more edges to be added
            n = edge[1]         # Otherwise get new vertex and
            vMap[n] = tree.nVertices() # map it into MST
            tree.addVertex(self.getVertex(n)) # copy it into MST
            tree.addEdge(        # Add weighted edge to MST mapping
                vMap[edge[0]], vMap[edge[1]], w) # vertex indices
        return tree            # Return the minimum spanning tree

    def weight(edge): return - edge[1] # Get neg. weight from edge tuple
```

Acknowledgments

We extend our deep appreciation to Andy Huber and Chantelle Marek for contributing to these errata.